1996

# A Graph-Constructive Approach to Solving Systems of Geomeric Constraints

Ioannis Fudos

Christoph M. Hoffmann
*Purdue University*, cmh@cs.purdue.edu

Report Number:
96-007

# A GRAPH-CONSTRUCTIVE APPROACH TO SOLVING SYSTEMS OF GEOMETRIC CONSTRAINTS

**Ioannis Fudos**
**Christoph M. Hoffmann**

**Department of Computer Science**
**Purdue University**
**West Lafayette, IN 47907**

**CSD TR-96-007**
**Janurary 1996**

# A Graph-constructive Approach to Solving Systems of Geometric Constraints*

Ioannis Fudos† Christoph M. Hoffmann

Department of Computer Science
Purdue University
West Lafayette, IN 47907-1398

Report CSD-TR-96-007
January 1996

**Abstract**

A graph-constructive approach to solving systems of geometric constraints capable of efficiently handling *well-constrained*, *overconstrained* and *underconstrained* configurations is presented. The geometric constraint solver works in two phases, in the *analysis phase* the constraint graph is analyzed and a sequence of elementary construction steps is derived, and then in the *construction phase* the sequence of construction steps is actually carried out. The analysis phase of the algorithm is described in detail, its correctness is proved, and an efficient algorithm to realize it is presented. The scope of the graph analysis is then extended by utilizing semantic information in the form of *angle derivations*, and by extending the repertoire of the construction steps. Finally the construction phase is briefly discussed.

*Work supported in part by ONR contract N00014-90-J-1599, and by NSF grants CDA 92-23502, ECD 88-03017, and CCR 95-05745.

†Supported in part by a Purdue Research Foundation Fellowship.

‡This and related reports are available via anonymous ftp to arthur.cs.purdue.edu under pub/cmh/Reports and subsidiaries.

# 1 Introduction

The problem of solving systems of geometric constraints is central to computer aided design and manufacturing [14, 25], stereochemistry [7], and robot motion planning. In CAD/CAM applications the user draws a sketch and annotates it with geometric constraints. Overconstrained and underconstrained configurations may occcur, deliberately or erroneously.

In *numerical constraint solvers* [31, 12, 23, 28, 20] the constraints are translated into a system of algebraic equations and are solved using iterative methods. To handle the exponential number of solutions and the large number of parameters iterative methods require sharp initial guesses. Most iterative methods have difficulties handling overconstrained or underconstrained instances. A sophisticated use of the Newton-Raphson method was developed in [20], where an improved way for finding the inverse Jacobian matrix is presented. When the Jacobian matrix is singular Light and Gossard use a modified version of Doolittle's method. *Rule-constructive solvers* [5, 2, 32, 30, 35, 34, 18] use rewrite rules for the discovery and execution of the construction steps. In this approach, complex constraints can be easily handled, and extensions to the scope of the method are straightforward to incorporate. Although it is a good approach for prototyping and experimentation, the extensive computations involved in the exhaustive searching and matching make it inappropriate for real world applications. A method is presented in [32], where handling of overconstrained and underconstrained problems is given special consideration. The *graph-constructive* [24, 19, 1] approach has two phases. During the first phase the graph of constraints is analyzed and a sequence of construction steps is derived. During the second phase these construction steps are followed to place the geometries. These approaches are fast and more methodical. [1] decomposes the system of geometric constraints in polynomial time in under-, over-, and well-constrained irreducible subsystems by an analysis of the bipartite graph of equations and variables. For a more extensive review the reader is referred to [10]

*Comparison with related work, specifically with rule constructive, numerical and owen on how they treat over and under.*

Figure 1 illustrates a case, where one of the four 90° angles and one of the two distances (length of $c$ and length of $a$) are redundant. However, the configuration is not well-constrained since the length of $b$ is not well specified or derivable.

The above considerations motivate the following technical contributions our paper makes:

- We extend our approach to solving systems of geometric constraints based on an analysis of the constraint graph that derives a sequence of elementary construction steps.

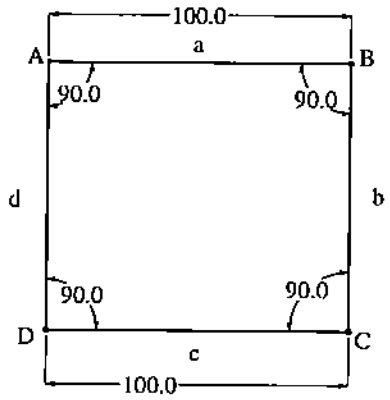


Figure 1: A system of geometric constraints defining a rectangle that is consistently overconstrained but is not well-determined since its height is not specified.

- The method can handle over-, under-, and well-constrained configurations.
- Efficient algorithms to analyze the constraint graph are presented.
- The method is formally studied as a rewrite system of sets and its correctness is proved. Note, that the behavior of the analysis for well-constrained problems has been reported before in [8].
- Extensions are presented that increase the scope of the core analysis techniques.
- The construction phase is briefly presented, its algorithmic compexity is studied and the problem of avoiding complex coordinates is discussed.

Section 2 provides an overview of our approach. To make the paper self-contained, Section 3 describes the reduction analysis for solving well-constrained and overconstrained configurations. The decomposition analysis for handling underconstrained configurations is presented in Section 4. Section 5 presents extensions that increase the scope of the core reduction analysis. Section 6 briefly describes the construction phase, derives its complexity and discusses the problem of solution with real number coordinates. Finally, Section 7 offers conclusions and future work.

## 2 An Overview

A geometric constraint problem is given by a set of points, lines, rays, circles with prescribed radii, line segments and circular arcs, called the *geometric*

*elements*, along with required relationships of incidence, distance, angle, parallelism, concentricity, tangency, and perpendicularity between any two geometric elements, called the *constraints*. As it is explained in [9] with an appropriate representation and some preprocessing we may restrict ourselves to points, lines, with pairwise distance and angle constraints. The geometric constraint problem is then formulated as follows:

> *Given a set $V$ of $n$ points and lines and a set $E$ of pairwise constraints among them, find an intuitive solution that satisfies the given constraints. A pairwise constraint may be one of: point-point distance, point-line distance and line-line angle. More formally $E$ is a partial mapping $E : V \times V \rightarrow \Re$.*

The problem can be coded as a *constraint graph* $G = (V, E)$, in which the graph nodes are the geometric elements and the constraints are the graph edges. The edges of the graph are labeled with the values of the distance and angle dimensions.

**Example 1** Figure 2 shows a dimensioned sketch defining a constraint problem involving 4 lines and 6 points. We have 8 implicit point-line distances that are 0, 2 explicit point-line distances, 3 angles and 4 point-point distances. Figure 3 shows the corresponding constraint graph. □

## 2.1 A Graph-constructive Method for Geometric Constraint Solving

Our constraint solving method first forms a number of rigid bodies with three degrees of freedom, called *clusters*. For simplicity we will assume that a maximum number of clusters is formed, each cluster consisting of exactly two geometric elements between which there exists a constraint. Geometrically, such a cluster corresponds to a pair of geometric elements whose position and orientation relative to each other is known.

Three clusters can be combined into a single cluster if they pairwise share a single geometric element. Geometrically, the combination corresponds to placing the associated geometric objects with respect to each other so that the given constraints can be satisfied.

The constraint solving method works in two conceptual phases:

- **Phase 1 (analysis phase):** The constraint graph is analyzed and a sequence of constructions is stipulated. Each step in this sequence corresponds to positioning three rigid geometric bodies (clusters) which pairwise share a geometric element (point or line).

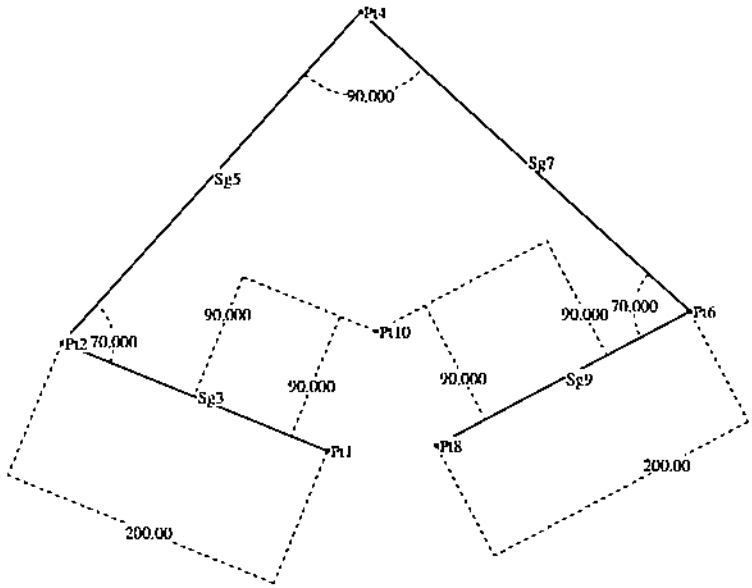


Figure 2: A well-constrained sketch defining a constraint problem with 10 geometric elements and 17 constraints.

- **Phase 2 (construction phase)**: The actual construction of the geometric elements is carried out, in the order determined by Phase 1, by solving certain standard sets of algebraic equations.

To illustrate the process, consider three points $A$, $B$, and $C$ between which distances have been prescribed, as shown in Figure 4 left. The associated con-

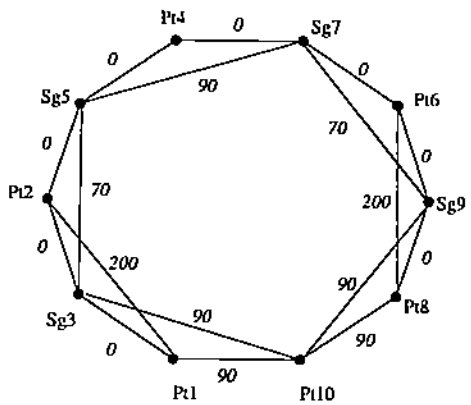


Figure 3: The constraint graph of the previous sketch

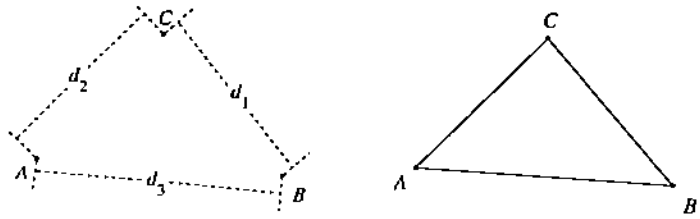


Figure 4: Constraint problem (left), and associated constrain graph (right)

straint graph is shown on the right. In Phase 1 of the constraint solving, we determine first that every pair of points can be constructed separately, resulting in three clusters. Moreover, the three clusters can be combined into a single cluster since they share pairwise a geometric element. The combination merges the three clusters into one. As soon as a single cluster is obtained, Phase 1 considers the constraint problem *solvable*.

Phase 1, the analysis phase, consists of two parts:

- the *reduction analysis* (Section 3) that produces a sequence of local cluster merges and handles well-constrained and overconstrained problems, and
- the *decomposition analysis* (Section 4) that produces a sequence of decompositions (that correspond to a reverse sequence of cluster merges) and handles underconstrained cases. The outcome of the reduction analysis is fed as input to the decomposition analysis.

Phase 2, the construction phase, is briefly discussed in Section 6.

## 2.2 Multiple Solutions and Root Identification

It is well known that a well-constrained geometric problem can have many incongruent solutions. Recall that at each construction step we may have to choose one of several solutions. Different choices may lead to incongruent solutions, each mathematically satisfying the given constraints.

In order to select a solution at each step, a number of heuristics are applied that make sense if the sketch with which the geometric problem has been specified has the same topological order type as the intended solution. This is an application-specific issue that is further discussed in [4].

We assume that the geometric problem has been specified by a user-prepared sketch. The point-line distances, and the angles between oriented lines are assumed to be signed quantities. The correct sign is determined from the original input sketch. Observing the sign conventions, all construction steps have a unique solution except in two cases, which are solved as follows:

(i) The relative placement of three points in a construction step has the same cyclic ordering in the plane as the ordering of the points in the original drawing.

(ii) The relative placement of two points and an oriented line is such that the inner product of the direction vector of the points and the line is sign invariant between the original sketch and in the chosen solution.

The geometric construction first places three geometric elements in this manner, and then applies a rigid-body transformation to align the three clusters accordingly. In particular, placing clusters by the shared geometric elements does not involve a reflection. We will prove later that no matter in which order the clusters are combined, the same set of triples is used to select the geometric solution, and that this implies congruence.

Note that the heuristics only imply the existence of a solution in a generic sense. Specific dimensions of distance and angle could be such that the solution selected by the heuristics would require complex coordinates. If this possibility is to be systematically excluded, some strategy would be required that searches the solution space in a canonical order. As the space of possible solutions may be exponential in the number of geometric elements, this is not an attractive prospect. In the case of ruler-constructible configurations, there is a theorem by Hilbert stating that if one solution has only real coordinates, then all of them must have real coordinates [13]. This means that for such configurations the heuristics will never fail to deliver a solution if one exists. The theorem does not generalize to ruler-and-compass constructible problems, and we know of no results that make progress beyond Hilbert's theorem. The problem of finding a real solution of a system of geometric constraints is further considered in Section 6.2.

## 2.3 Well-constrained, Overconstrained and Underconstrained Problems

Each line or point on the Euclidean plane has two degrees of freedom. Each distance or angle corresponds to one equation. If we have no fixed geometric elements (geometric elements whose absolute coordinates have been specified explicitly by the user) then we expect that

$$|E| = 2|V| - 3, \text{ where } |V| = n$$

Recall that $|V|$ is the number of geometric elements and that $|E|$ is the number of constraints. Note that the solution will be a rigid body with three remaining degrees of freedom, because the constraints determine only the relative position of the geometric elements. We use this argument to define a technical notion of well-constrained sketches in which no attempt is made to account for the

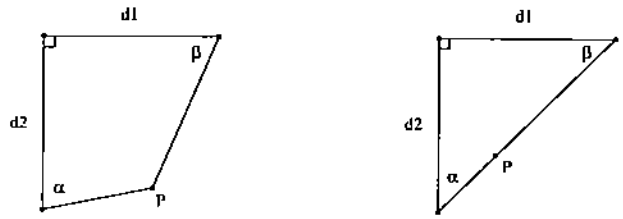


Figure 5: Degenerate Configuration (right) for $\alpha + \beta = 90°$.

possibility that for special dimension values an otherwise well-constrained problem may happen to be underconstrained. An example is shown in Figure 5. In the figure, the vertex $P$ of the quadrilateral has a well-defined position when $\alpha + \beta \neq 90°$. But for $\alpha + \beta = 90°$ the position of $P$ is not determined. This "semantic" notion of well-constrained problems is too specific for the constraint graph analysis because there the generic problem of constructing a solution is considered independent of dimension values.

Intuitively a dimensioned sketch is considered to be well constrained if it has a finite number of solutions for nondegenerate configurations. Similarly a dimensioned sketch is considered to be underconstrained if it has an infinite number of solutions for nondegenerate configurations. Finally a dimensioned sketch is considered to be overconstrained if it has no solutions for nondegenerate configurations.

The intuitive notions above can be made technically precise as follows:

**Definition 1** A graph with $n$ nodes is *structurally overconstrained* if there is an induced subgraph with $m \leq n$ nodes and more than $2m - 3$ edges.

**Definition 2** A graph is *structurally underconstrained* if it is not overconstrained, and the number of edges is less than $2n - 3$.

**Definition 3** A graph is *structurally well-constrained* if it is not overconstrained, and the number of edges is equal to $2n - 3$.

**Definition 4** A geometric constraint problem with a structurally over-, under- or well-constrained constraint graph is called a structurally over-, under- or well-constrained problem, respectively.

For an algorithm to test whether a graph is structurally well-constrained see, e.g. [16, 29]. Note that a structurally well-constrained graph can be overconstrained in a geometric sense, for example if there are three lines with pairwise angle constraints.

The core reduction analysis handles structurally well-constrained and overconstrained problems. Section 3 presents this method in detail, together with a correctness proof and an efficient algorithm to realize it. Section 4 presents the decomposition analysis that handles structurally underconstrained problems.

# 3 The Reduction Analysis

We are given a constraint graph $G = (V, E)$ whose nodes $V$ are geometric elements, and whose edges $E$ are the geometric constraints. Without loss of generality, the geometric elements consist only of points and lines, and the constraints are only of distance and angle.

We consider sets $\mathbf{C}$ whose elements are sets $S$ that in turn have as elements nodes of $G$. Each set $S$ represents a cluster. Intuitively, a cluster $S$ consists of geometric elements whose position relative to each other has already been determined. A cluster thus can be considered a rigid geometric structure that has three degrees of freedom, two translational and one rotational.

Initially, we form a set $\mathbf{C}_G$ from $G$. For each edge $e = (u, v)$ in $G$, there is a cluster $S_e = \{u, v\}$. The construction steps that solve the constraint problem amount to one *reduction step* that merges three clusters whose pairwise intersection is a singleton. The reduction is denoted by $\rightarrow$. The process of finding a sequence of reductions that derives a single set of clusters, and thus determines, a sequence of construction steps that positions the geometric elements to satisfy the initial set of constraints, is called *reduction analysis*.

**Example 2** To illustrate better the process of the reduction analysis, consider the constraint graph of Figure 3. After detecting a sequence of cluster merges, we end up with three clusters, **U**, **V** and **W**, as shown in Figure 6. The analysis concludes by merging the three clusters into one. □

## 3.1 Correctness

In [8], we considered clusters as sets and studied their structure under reduction. Given the constraint graph $G = (V, E)$, we define the set of clusters

$$\mathbf{C}_G = \{\{u, v\} : (u, v) \in E\}$$

Cluster sets are rewritten using a reduction $\rightarrow$. The reduction $\rightarrow$ is formally defined as follows:

**Definition 5** Let $\mathbf{C}$ be a set of clusters $\mathbf{C}$ in which there are three clusters $S_1$, $S_2$, $S_3$ such that

$$\begin{aligned} S_1 \cap S_2 &= \{g_1\} \\ S_2 \cap S_3 &= \{g_2\} \\ S_3 \cap S_1 &= \{g_3\} \end{aligned}$$

where $g_1$, $g_2$, $g_3$ are distinct, then

$$\mathbf{C} \rightarrow \mathbf{C}'$$

where

$$\mathbf{C}' = (\mathbf{C} \cup \{S_1 \cup S_2 \cup S_3\}) - \{S_1, S_2, S_3\}$$

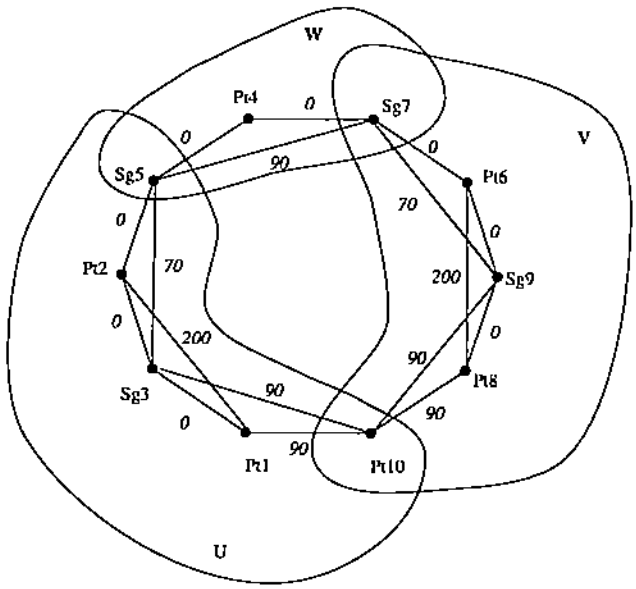


Figure 6: Finding the clusters of the graph

We proved first a weak notion of correctness:

> If the constraint graph is not structurally overconstrained, then our method reduces the initial set $\mathbf{C}_G$ to the same (irreducible) normal form, no matter in which order the reduction steps are applied. That is the set $\mathbf{C}_G$ and the reduction $\rightarrow$ are a terminating, confluent rewriting system (see e.g., [26]).

Here, confluent means that if a set $A$ can be reduced to two different sets $B_1$ and $B_2$, then there are two reduction sequences, one reducing $B_1$, the other $B_2$ to the same set $C$.

Notice, however, that a well-constrained geometric problem has in general several incongruent solutions (see Section 2.2). In [8], we proved therefore a stronger uniqueness theorem:

> If the constraint graph is well-constrained and our algorithm reduces the initial set $\mathbf{C}_G$ to a single cluster using, in the construction phase, the placement rules given in Section 2.2, then the solutions derived by different reduction sequences place a fixed set of triples of geometric elements in the same relative position.

This result implies that different reduction sequences must produce geometric solutions that are congruent.

## 3.2 An Efficient Reduction Algorithm

In this section, we provide an algorithm that runs in time quadratic in the number of geometric elements and constraints and realizes correctly the reduction analysis.

Let $G = (V, E)$ be the constraint graph. Let $n = |V|$ and $e = |E|$. The algorithm has a $O(n^2)$ worst case time complexity for constraint graphs that are not structurally overconstrained. We can also show that we can test, in the same time bound, whether the graph is structurally overconstrained (see Section 3.3). For the purposes of the algorithm we will also consider an undirected *cluster graph* $H$ whose vertices are the edges and vertices of $G$. $H$ has an edge $(e, v)$ iff there is an edge $e \in E$ and $e$ is incident to $v$ in $G$. Note that $H$ is bipartite.

The initial cluster graph $H$ records of clusters of size 2 in $G$. $H$ is bipartite, with one set of vertices corresponding to clusters, initially the edges of $G$, the other corresponding to the geometric elements of the constraint problem, the vertices of $G$. There is an edge in $H$ if a vertex of $G$ belongs to a cluster.

In this section we will assume that $G$ is not structurally overconstrained thus every subgraph $G_s = (V_s, E_s)$ of $G$ satisfies $|E_s| \leq 2|G_s| - 3$.

### 3.2.1 Overview

The algorithm for solving the constraint graph is as follows:

1. Construct initial clusters of size 2, each consisting of two adjacent vertices of $G$.

2. Construct the cluster graph $H$.

3. Find all triangles in $G$.

4. Successively rewrite $H$ by replacing a 6-cycle in $H$ by a four-node structure as explained below. Record a cluster merging operation for each such rewriting step.

5. If $H$ can be rewritten into a final graph that is a star with center a cluster and periphery the vertices of $G$, then $G$ is solvable; otherwise it is not solvable.

A 6-cycle in $H$ corresponds to three clusters that pairwise share an element (a vertex of $G$). The rewriting step corresponds to a cluster merge. Let $(u, U, v, V, w, W)$ be such a 6-cycle, where $u, v, w \in G$. We replace the three vertices $U$, $V$ and $W$ with a new vertex $X$. Then $X$ will be incident to all vertices that $U$, $V$ and $W$ are adjacent to. That is, the nodes $U, V, W$ are combined into a single node. See also Figure 7.

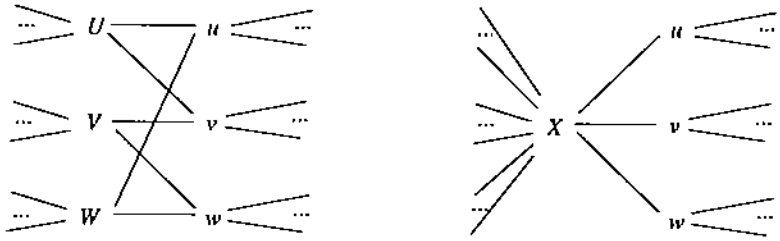


Figure 7: Rewriting a 6-Cycle in the Cluster Graph

A 4-cycle in $H$ corresponds to two clusters that share two elements. If a 4-cycle exists, the graph $G$ is structurally overconstrained. Thus, the shortest possible cycle in $H$ has length 6.

### 3.2.2 Details

The main work of the algorithm is to find and reduce 6-cycles. Finding triangles in Step 3 identifies all 6-cycles in $H$, and the algorithm will find other 6-cycles that are formed by rewriting in Step 4. Those two steps must be implemented carefully, and we explain how they are done.

*Step 3:* We assume that $G$ is represented both by adjacency lists and by the adjacency matrix. We find all triangles in $G$, using the method of [17]:

Build a depth-first search tree. Three types of triangles are possible, involving two, one, or no tree edge; Figure 8. Triangles that involve one or two tree

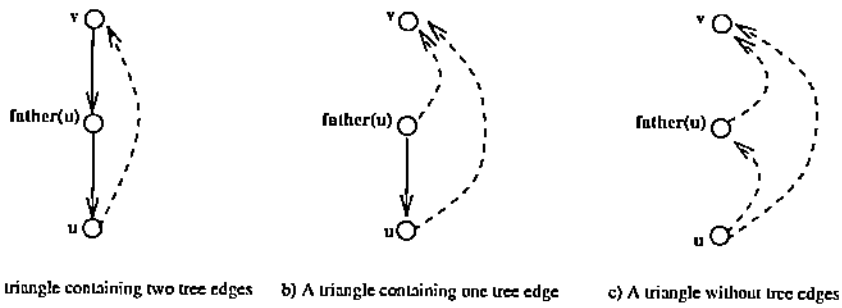


Figure 8: Three cases for a triangle in a depth first search

edges are found uniformly as follows: Let $(u, v)$ be a back edge. If $(father(u), v)$ is an edge in $G$, we have a triangle with one or two tree edges, because in that case either $(v, father(u))$ is a tree edge or $(father(u), v)$ is a back edge.

Next, we remove all tree edges, and repeat the above search for all connected components of the remaining graph. This is repeated until there are no more edges. Clearly, all triangles that do not have tree edges in the first depth-

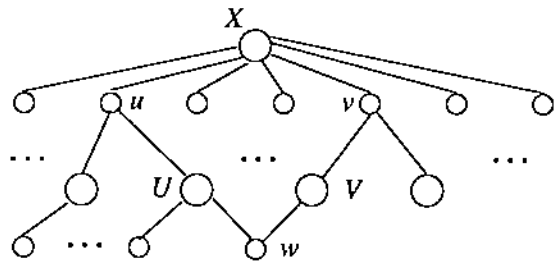


Figure 9: Finding a 6-cycle by doing a breadth first search

first search eventually become triangles involving tree edges in later depth-first searches.

*Step 4:* A triangle in the constraint graph $G$ corresponds to a 6-cycle in the cluster graph $H$ and vice versa. Having found all triangles in $G$, we now know all six cycles of $H$.

When rewriting a 6-cycle, new 6-cycles could be created. They are found by a limited-depth breadth-first search that originates at the new vertex $X$; see also Figure 9. In [22] limited depth-first search has been used to find cycles of minimum length. To find all such 6-cycles, begin a breadth-first search at $X$. Each vertex at level 3 that is reached twice in the breadth-first search closes a 6-cycle. Note that the search is done only to level 3.

### 3.2.3 Time Analysis

Steps 1 and 2 require $O(e)$ steps in all. Since $G$ is not structurally overconstrained, this is linear in the number of vertices $n$.

Step 3 repeatedly performs depth-first search. It requires the adjacency matrix of $G$ which costs $O(n^2)$ if constructed with standard data structures. The first depth-first search now requires $O(e)$ steps, because the test whether $(father(u), v)$ is a graph edge can be done in constant time using the adjacency matrix.

Let $G_1$ be a connected component of $G$ with $m$ vertices. After a depth-first search of $G_1$, $m-1$ edges are removed from the adjacency lists. This is more than half the edges in $G_1$ because $G$ is structurally not overconstrained. Consequently, the next depth-first search examines less than half the number of edges. The total time for Step 3 is therefore $O(e)$ which is $O(n)$, excluding the time for the adjacency matrix construction.

In Step 4, each reduction of a 6-cycle takes time $O(e)$: We go through all the adjacency lists and replace each of the three vertices $U$, $V$ and $W$ with vertex $X$. Every new 6-cycle must involve the new node $X$. We do a breadth-first search beginning at $X$ to depth 3, looking for new 6-cycles. This again takes again

time $O(e)$. Initially, $H$ contains at most $2n-3$ vertices representing clusters. Each 6-cycle replacement reduces this number by 2, thus in Step 4 we may have at most $n$ reductions. And since each breadth first search corresponds to one reduction, Step 4 takes time $O(ne) = O(n^2)$.

### 3.3 Structurally Overconstrained Problems

We first show that we can test, in the same time bound as before, whether the graph is structurally overconstrained.

With some extra checking the algorithm of Section 3.2 can detect an overconstrained subgraph in the same time bound. For Step 1 before doing a depth first search in a connected component graph $G_1 = (V_1, E_1)$ we check whether $|E_1| \leq 2|V_1| - 3$, if yes we go on, otherwise we terminate the algorithm and return the graph $G_1$. Since this step is performed on the initial graph as well we have ensured that the original graph has less than $2n-3$ edges. Step 2 takes again time $O(n)$, and Step 3 takes time $O(n^2)$. For Step 4, we keep reducing until we meet a 4-cycle, then we terminate the algorithm and return the 4-cycle, the time complexity is again $O(n^2)$. The returned graph is used for interactive editing.

To handle consistently overconstrained problems we introduce a new reduction operation that merges two clusters sharing two or more geometric elements. This corresponds to checking in the construction phase whether the relative positioning of the shared geometric elements in the two clusters is consistent. If it is consistent, then the two rigid bodies are merged into one. A cluster configuration derived from a structurally overconstrained problem is depicted in Figure 10. In this cluster configuration, $C_1$, $C_2$ and $C_3$ are merged into a cluster $C'$, and $C_4$, $C_5$ and $C_6$ into a cluster $C''$. $C'$ and $C''$ have two common elements: $p_4$ and $p_2$. In the analysis phase we merge the two clusters and in the construction phase we first check whether the distance between $p_4$ and $p_2$ in $C'$ matches the distance between $p_4$ and $p_2$ in $C''$. If so, we rotate and translate, e.g. $C'$ to match $C''$. We can prove that by adding this new reduction, we get a confluent rewrite systems for all cluster configurations (overconstrained or nonoverconstrained). However, congruence cannot be proved by the techniques of [8], since different reduction choices will result in different sets of geometry triples.

We assume that in the initial constraint graph there may be no more than one edge between any two vertices. Otherwise we can find such multiple edges in time linear to the total number of edges and reduce them. Then the only modification to the algorithm of Section 3.2 is to add to Step 4, that if we meet a 4-cycle in a breadth first search we reduce it as if it was a 6-cycle and we go on. To derive the time bounds note that the number of reductions is always linear in the number of nodes in $H$. Also recall that each iteration of Step 4 takes only $O(e)$. Steps 1 and 2 take time $O(e)$. Step 3 becomes $O(e\sqrt{e})$ (see [17]), and Step 4 becomes $O(e^2)$. This gives an overall $O(e^2)$ worst case time

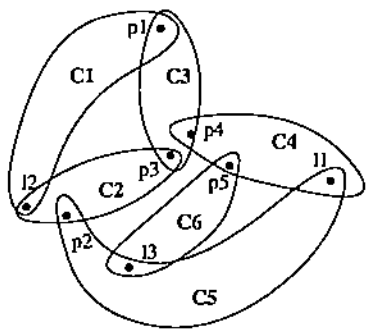


Figure 10: A cluster configuration derived from a structurally overconstrained problem.

complexity for the general case.

If the original graph has $e = O(n)$ then Steps 1 and 2 take $O(n)$ time, Step 3 takes $O(n\sqrt{n})$ and Step 4 takes time $O(n^2)$.

# 4 The Decomposition Analysis

The bottom-up reduction analysis of Section 3 works well with overconstrained and well-constrained problems. However, it has difficulties when dealing with underconstrained problems, because there appears to be no reliable way to locally add constraints deduced from the input sketch to transform an underconstrained problem to a well-constrained one.

**Example 3** Figure 11 shows the constraint graph of an underconstrained geometric problem, that needs the addition of three constraints to become well-constrained and solvable. The $v_k$ in the graph represent points, and the edges distance constraints between them. Adding distance constraints between $(v_1, v_3)$, $(v_7, v_9)$ and $(v_{13}, v_{15})$ make the problem well-constrained and solvable. However, if we add a distance constraint for $(v_{16}, v_{10})$ to trigger a local cluster merging, then any addition of two more distance constraints will make the problem nonsolvable. Thus the local reduction analysis is insufficient for underconstrained problems. A global analysis is needed. □

We present now a global, top-down *decomposition analysis* that does especially well for underconstrained problems. The decomposition analysis also handles well-constrained problems efficiently, but does not do well on overconstrained problems, nor does it make it easy to include the angle transformations, described in Section 5.2. Note that a top-down decomposition analysis was first proposed by Owen [24]. His algorithm runs in quadratic time and uses the linear

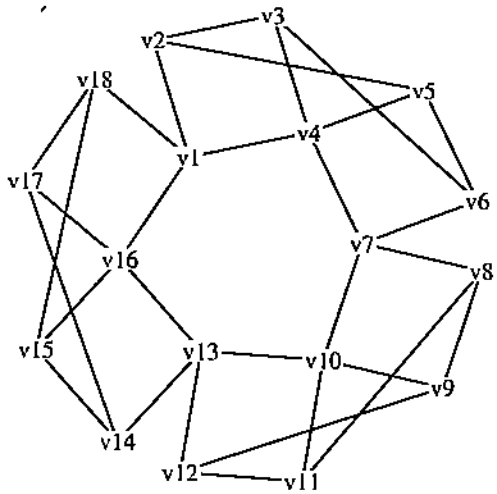


Figure 11: A structurally underconstrained graph where adding edges (clusters) to trigger local cluster merging does not always work.

algorithm for finding triconnected components presented in [15].

The decomposition analysis presented now analyzes the cluster configuration derived by the reduction analysis and is proved to handle all underconstrained problems that can become well-constrained and solvable by adding constraints. Our decomposition algorithm has quadratic worst case time complexity and uses the classical linear depth-first search algorithms for finding split components and articulation nodes [15, 33].

## 4.1 A Conceptual Algorithm

Initially, all clusters found by the reduction analysis are considered to be in a set S. The set S is partitioned into two or three subsets $S_k$. Let $S_1$ and $S_2$ be two such subsets. We require that there is at most one geometric element shared by the clusters in the two sets. That is, let $G_1 = \{g \mid g \in C, C \in S_1\}$ be the geometric elements that are in clusters of the set $S_1$, and let $G_2 = \{g \mid g \in C, C \in S_2\}$. Then we require that $|G_1 \cap G_2| \leq 1$. At each decomposition step, we so subdivide a set of clusters S into two or three disjoint cluster sets. The elementary decomposition steps are shown in Figure 12. In the figure, $S_i$, denotes a set of clusters, and $C_i$ denotes an individual cluster. A decomposition of type (a) is a simple set partition. In the case of a decomposition of type (b), a new cluster $C_1$ is created, corresponding to adding a constraint. A decomposition step of type (c) creates three additional clusters. The corresponding added constraints

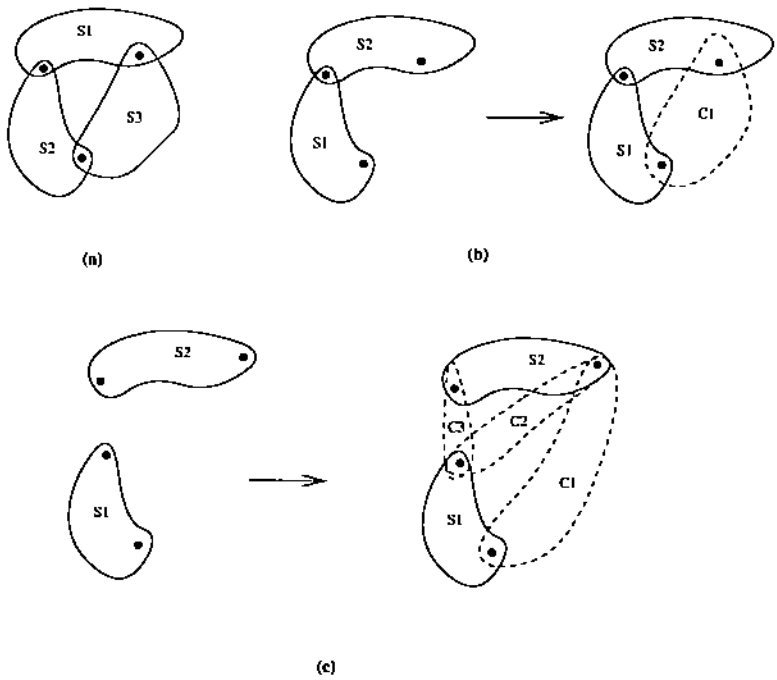


Figure 12: The three basic cases for decomposing the clusters into sets.

can be formulated from the relative position of the geometric elements in the input sketch. Decomposition continues until every cluster of the original set S is in a singleton set $S_k$.

A decomposition step corresponds to a cluster merge in cases (a) and (b), and to two cluster merges in case (c). These cluster merge operations will be executed in reverse order of the decomposition and are done when a single cluster is obtained for the entire graph, corresponding to the set S with which decomposition began.

We avoid selecting a partition where all three shared geometric elements are lines in case (a). In case (b), when the shared element is a line we choose to add a cluster $C_1$ that does not consist of two lines. The restriction avoids a corresponding merge of three clusters with lines as the shared geometric elements, for such a merge is geometrically undetermined. A similar restriction is placed on the clusters $C_1$, $C_2$ and $C_3$ in decompositions of type (c).

In case that $S_1$ and $S_2$ are singleton sets whose clusters consist of lines only, then a virtual point is added as shown in Figure 13.

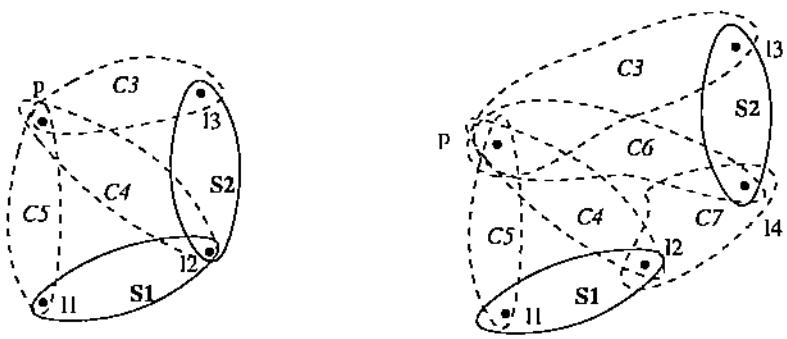


Figure 13: Decompositions (b) and (c) for clusters containing only lines. A virtual point $p$ is added plus constraints to form the clusters $C_k$.

The reduction analysis has no effect on the underconstrained graph of Figure 11 because there are no triangles. The decomposition analysis, however, can successfully solve the underconstrained problem and identify three additional constraints that will make it solvable. For instance, the clusters formed initially (one for each edge) can be split into three sets that share geometric elements $v_1$, $v_7$ and $v_{13}$.

Decomposition cannot handle problems that are consistently overconstrained, such as the one shown in Figure 10. There is no decomposition of the clusters into two or three sets that pairwise share at most one geometric element. However, the reduction analysis can efficiently handle such geometric constraint

problems.

### 4.2 Correctness

Let $\mathcal{C}$ be a set of sets $\mathbf{S}$. Each set $\mathbf{S}$ is a set of clusters $C$. We assume that the cluster sets $\mathbf{S}$ of $\mathcal{C}$ are disjoint. Initially, $\mathcal{C}$ contains only one set whose elements are the clusters derived by the reduction analysis.

The type (a) decomposition of Figure 12 is denoted $\rightarrow^{D(a)}$, and is formally defined as follows:

**Definition 6** Let $\mathcal{C}$ be a collection of sets of clusters in which there is a set of clusters $\mathbf{S}$ whose elements can be partitioned in three nonempty sets $\mathbf{S}_1$, $\mathbf{S}_2$ and $\mathbf{S}_3$ such that:

$$\begin{aligned}(\bigcup_{C\in\mathbf{S}_1} C)\cap(\bigcup_{C\in\mathbf{S}_2} C) &= \{g_1\}\\ (\bigcup_{C\in\mathbf{S}_2} C)\cap(\bigcup_{C\in\mathbf{S}_3} C) &= \{g_2\}\\ (\bigcup_{C\in\mathbf{S}_3} C)\cap(\bigcup_{C\in\mathbf{S}_1} C) &= \{g_3\}\end{aligned}$$

where $g_1$, $g_2$, $g_3$ are distinct and are not all lines, then

$$\mathcal{C} \rightarrow^{D(a)} \mathcal{C}'$$

where

$$\mathcal{C}' = (\mathcal{C} \cup \{\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3\}) - \{\mathbf{S}\}$$

The type (b) decomposition of Figure 12, denoted $\rightarrow^{D(b)}$, is formally defined as follows:

**Definition 7** Let $\mathcal{C}$ be a collection of sets of clusters in which there is a set of clusters $\mathbf{S}$ whose elements can be partitioned in two nonempty sets $\mathbf{S}_1$ and $\mathbf{S}_2$ such that:

$$(\bigcup_{C\in\mathbf{S}_1} C)\cap(\bigcup_{C\in\mathbf{S}_2} C) = \{g\}$$

then

$$\mathcal{C} \rightarrow^{D(b)} \mathcal{C}'$$

where

$$\mathcal{C}' = (\mathcal{C} \cup \{\mathbf{S}_1, \mathbf{S}_2\}) - \{\mathbf{S}\}$$

Finally, the type (c) decomposition of Figure 12 which we denote by $\rightarrow^{D(c)}$ is formally defined as follows:

**Definition 8** Let $\mathcal{C}$ be a collection of sets of clusters in which there is a set of clusters $\mathbf{S}$ whose elements can be partitioned in two nonempty sets $\mathbf{S}_1$ and $\mathbf{S}_2$ such that:

$$(\bigcup_{C\in\mathbf{S}_1} C)\cap(\bigcup_{C\in\mathbf{S}_2} C) = \emptyset$$

then,

$$\mathcal{C} \rightarrow^{D(c)} \mathcal{C}'$$

where

$$\mathcal{C}' = (\mathcal{C} \cup \{S_1, S_2\}) - \{S\}$$

**Definition 9** A decomposition $\rightarrow^D$ applied to a collection $\mathcal{C}$ is one of $\rightarrow^{D(a)}$, $\rightarrow^{D(b)}$, or $\rightarrow^{D(c)}$.

We will show that the rewrite system $(C_G, \rightarrow^D)$ has a unique normal form that is obtained after finitely many steps.

**Definition 10** The collection $\mathcal{C}'$ of cluster sets is *derived* from $\mathcal{C}$, if $\mathcal{C}'$ can be obtained by applying a finite sequence $\tau$ of decompositions to $\mathcal{C}$. We denote this by

$$\mathcal{C} \rightarrow_*^D \mathcal{C}'$$

**Definition 11** A collection $\mathcal{C}$ of cluster sets to which $\rightarrow^D$ is not applicable is called *nondecomposable*. If $\mathcal{C}$ is nondecomposable and can be derived from $\mathcal{C}'$, then $\mathcal{C}$ is called a *normal form* of $\mathcal{C}'$. A collection of cluster sets whose normal form contains only singleton cluster sets is called *solvable*.

We will show that a collection of cluster sets has a unique normal form and is derived by a finite sequence of decomposition steps.

**Lemma 1** A normal form of a collection of cluster sets is derived by a decomposition sequence whose length is bounded by $c - 1$ where $c$ is the total number of clusters.

**Proof**

(By induction on the number of clusters). For $c = 1$, $\mathcal{C} = \{S\}$ and $S = \{C\}$, which is already in normal form. Assume the lemma holds for all sets with fewer than $c > 1$ clusters, and let $\mathcal{C}$ be such that there are $c$ clusters in all in the element sets of $\mathcal{C}$. If $\mathcal{C}$ has more than 1 element, then each set $S \in \mathcal{C}$ has fewer than $c$ elements. By the induction hypothesis, $\mathcal{C}$ can be brought into normal form in at most $c - 1$ steps. Otherwise, $\mathcal{C} = \{S\}$. If $\mathcal{C}$ is not in normal form already, then $S$ can be decomposed in one step into two or three sets $S_k$, each with fewer than $c$ elements. By the induction hypothesis, therefore, normal form is reached in at most $1 + (c_1 - 1) + (c_2 - 1) = c - 1$ steps or in at most $1 + (c_1 - 1) + (c_2 - 1) + (c_3 - 1) = c - 2$ steps. □

**Theorem 1** The rewrite system $(\mathcal{C}, \rightarrow^D)$ is confluent.

**Proof**

Let $\mathcal{C}$ be a collection of cluster sets. Assume that two different decomposition steps , $\mathcal{C} \rightarrow_1^D \mathcal{C}_1$ and $\mathcal{C} \rightarrow_2^D \mathcal{C}_2$, are applicable to $\mathcal{C}$. We will prove that $\mathcal{C}_1 \rightarrow_*^D \mathcal{C}'$ and $\mathcal{C}_2 \rightarrow_*^D \mathcal{C}'$.

If $\rightarrow_1^D$ and $\rightarrow_2^D$ partition two different sets $S_1$ and $S_2$ of $\mathcal{C}$, then $\mathcal{C}_2 \rightarrow_1^D \mathcal{C}'$ and $\mathcal{C}_1 \rightarrow_2^D \mathcal{C}'$. So, we assume that $\rightarrow_1^D$ and $\rightarrow_2^D$ partition the same set $S$ of clusters. Without loss of generality we assume that $\mathcal{C} = \{S\}$.

Assume that $\rightarrow_1^D$ and $\rightarrow_2^D$ are both of type (a) and partition S into $S_{11}$, $S_{12}$, $S_{13}$ and into $S_{21}$, $S_{22}$, $S_{23}$, respectively. Intuitively, the collection $C'$ is $C' = \{S'_{ik}\}$, where $S'_{ik} = S_{1i} \cap S_{2k}$, $1 \leq i, k \leq 3$ and $S'_{ik} \neq \emptyset$.

We want to decompose $S_{11}$ into the sets $S'_{11}$, $S'_{12}$ and $S'_{13}$, assuming that all three sets are not empty. Since the decomposition $\rightarrow_2^D$ requires that $S_{21}$, $S_{22}$ and $S_{23}$ pairwise share one geometric element, the sets $S'_{11}$, $S'_{12}$ and $S'_{13}$ pairwise share at most one geometric element. Therefore, the decomposition of $S_{11}$ can be done by in one of four ways, depending on the number of shared geometric elements:

(3) — by a single step of type (a)
(2) — by two steps of type (b)
(1) — by one step of type (b) and one step of type (c)
(0) — by two steps of type (c)

If one or more of the $S'_{1k}$ are empty, a similar case analysis establishes the decomposition of $S_{11}$. By symmetry, $S_{12}$ and $S_{13}$ are decomposed into the sets $S'_{2k}$ and $S'_{3k}$. Therefore,

$$C \rightarrow_1^D C_1 \rightarrow_*^D C'$$

if both reductions are of type (a). Again, by symmetry, we also have

$$C \rightarrow_2^D C_2 \rightarrow_*^D C'$$

It is now routine to argue confluence in the cases where one, or both, of $\rightarrow_1^D$ and $\rightarrow_2^D$ are of type (b) or type (c). □

**Corollary 1** (Normal Form Theorem)
A collection $C$ has a unique normal form under $\rightarrow^D$ that is obtained by finitely many decomposition steps.

**Proof**
Immediate from Theorem 1 and Lemma 1. □

Note that two decomposition sequences need not result in congruent geometric solutions, since different decompositions may determine different choices of constraint additions. An example is shown in Figure 14. The user specifies the underconstrained problem in (a). Assume that the distance between Pt1 and Pt2 is 13.738, and between Pt5 and Pt6 is 17.000, in the input sketch. Then, depending on the decomposition sequence, solutions (b) or (c) may be obtained.

## 4.3 An Efficient Algorithm

The conceptual decomposition method just described can be implemented to run in time quadratic in the number of clusters and geometric elements. It can also be combined with the reduction analysis of Section 3 because the decomposition only requires abstract clusters that are sets of geometric elements whose relative

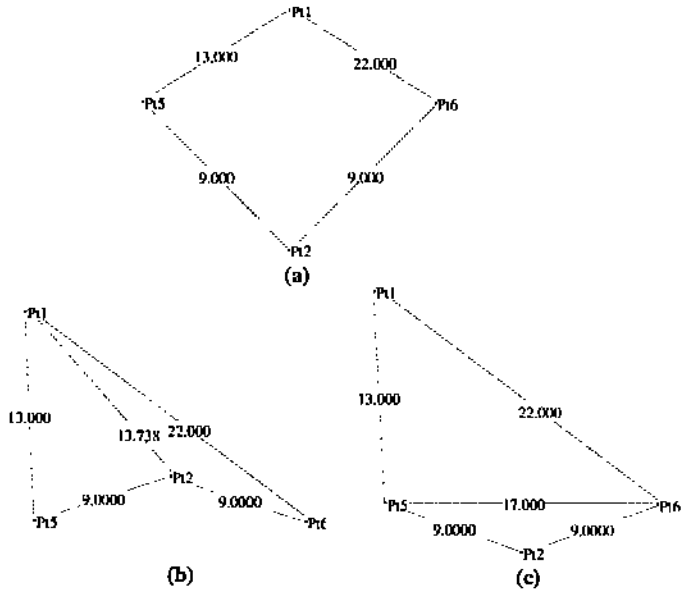


Figure 14: Getting two geometrically incongruent solutions, (b) and (c), from the same underconstrained draft (a).

position is known. Thus, the set of clusters obtained by the reduction analysis is the initial set S of the decomposition analysis.

For the implementation of decomposition we will make use of the graph $G_R = (V_R, E_R)$ derived by the reduction analysis. Recall that this graph is bipartite, and the partition is $V_R = V_R^c \cup V_R^g$, where $V_R^c$ are the clusters and $V_R^g$ are the geometric elements. The graph edge $(C, g)$ means that the geometric element $g$ belongs to the cluster $C$.

We construct a graph $G_D = (V_D, E_D)$ from the graph $G_R = (V_R, E_R)$, where

$$V_D = V_R^g \cup \{v^1, v^2, v^3 \mid v \in G_R^c\}$$

consists of the nodes of $V_R^g$ plus three nodes $v^k$ for every node $v$ in $V_R^c$. The edges of $G_D$ are

$$E_D = \{(u, v^1), (u, v^2), (u, v^3) \mid (u, v) \in E_R,\ u \in G_R^g,\ v \in G_R^c\}$$

We will use a DFS algorithm for finding the split components of a biconnected graph. We recall some terminology from [15].

Let $G$ be a connected undirected graph. Then $a$ is an *articulation node* of $G$ iff there are two vertices $u$ and $v$ different from $a$ such that $a$ is on every path connecting $u$ and $v$. A graph with no articulation nodes is called *biconnected*. Let $a$ and $b$ be two vertices in a biconnected graph $G$. The edges of $G$ are divided

into *separation classes* $E_1, E_2, \ldots E_k$. Two edges are in the same separation class $E$ if there is a path using both edges and not containing $a$ or $b$ except, possibly, as endpoints.

If the two vertices $a$ and $b$ divide the edges into more than two separation classes, then $\{a, b\}$ is an *articulation pair* of $G$. Moreover, if $\{a, b\}$ divides the edges into two separation classes, each containing more than one edge, then $\{a, b\}$ is also an articulation pair. If $G$ has no articulation pairs then $G$ is *triconnected.*

Assume there is an articulation pair $\{a, b\}$ that induces the separation classes $E_1, E_2, \ldots E_k$. Let $E' = \bigcup_{i=1}^{m} E_i$ and $E'' = \bigcup_{i=m+1}^{k} E_i$, such that $|E'|, |E''| \geq 2$, and let $G' = (V(E'), E' \cup \{(a, b)\})$ and $G'' = (V(E''), E'' \cup \{(a, b)\})$. Then the graphs $G'$ and $G''$ are called *split graphs* of $G$. In this case we say that $G$ has been split into $G'$ and $G''$. The added edge $(a, b)$ is labeled to denote the split and is called a *virtual bond.* By *merging* $G'$ and $G''$ we mean the reconstruction of $G$ from the two graphs.

Assume that we recursively split $G$ and its split graphs until we obtain graphs that cannot be split further. The set of these (triconnected) graphs is called a set of *split components* of $G$. The decomposition of a biconnected graph $G$ into two split components is not unique because the partition of the separation classes is arbitrary. By merging the split components pairwise we recover the original graph. We distinguish three types of split components: triple bonds, triangles and other triconnected graphs.

If we perform all merging operations between bonds and all merging operations between triangles, then the resulting graphs will be bonds and polygons plus the nontrivial triconnected split components. It is a standard result that the decomposition of a graph $G$ into triconnected components is unique [21].

The algorithm presented in this section is based on the following lemma.

**Lemma 2** A cluster configuration with a biconnected graph $G_D$ as defined above, is type (a) decomposable if and only if every set of split components of the graph contains at least one triangle whose edges are all virtual bonds, and whose vertices are not all lines.

**Proof**

Assume that among the split components of $G_D$ there is a triangle $(a, b, c)$ whose edges are virtual bonds, and whose vertices are not all lines. Then, since we have replicated each cluster node and all the adjacent edges, we are certain that no articulation node or articulation pair will contain any cluster nodes. Thus, all three articulation pairs $(a, b)$, $(b, c)$ and $(c, a)$ correspond to geometric elements, and $G_D$ can be decomposed into three sets of clusters, sharing pairwise a single geometric element. These sets can be found if we follow the labels of the virtual bonds (see Figure 16).

Now assume that the cluster configuration can be decomposed into three sets

of clusters pairwise sharing exactly one geometric element. Let $(a, b, c)$ be the shared geometric elements, and assume without loss of generality that vertex $a$ corresponds to a point. We can find at least one decomposition of $G_D$ into split components containing the triangle: $(\{a, b, c\}, \{(a, b), (b, c), (c, a)\})$. We know however (see e.g. [21]), that the decomposition of any graph in triconnected components is unique. We also know from [15] that merging split components that are triangles as much as possible, will give us a number of polygons which are triconnected components of the original graph. Since polygons occur as triconnected components only as a result of two or more triangles being merged (otherwise they would split), we conclude that every decomposition of $G_D$ into split components will contain a triangle graph where one vertex is $a$ and the edges are virtual bonds (there are no edges between geometric elements and all articulation pairs consist of geometric elements). □

By the above lemma and by observing that a cluster node cannot be an articulation node, we can show that the following algorithm correctly implements the decomposition analysis described earlier.

1. If the graph is not connected then split it into connected components. If the graph has $l$ connected components then there are $l - 1$ decompositions of type (c) that split the original graph. Apply Step 2 to each of the connected components.

2. If the graph is not biconnected then split it into biconnected components. If the graph has $k$ biconnected components, then $k - 1$ decompositions of type (b) split the graph into those components. Apply Step 3 to each of the biconnected components.

3. Find all split components of the graph. If there is no articulation pair stop.

   If among the split components there is a triangle whose edges are virtual bonds and whose vertices are not all lines, then the overall graph is type (a) decomposable and the separation triple consists of the vertices of the triangle. Find the three component graphs by following the virtual bonds. Then apply the algorithm recursively to the three components.

When the algorithm terminates, every cluster represents a triconnected component of the constraint graph that cannot be split further.

Figure 15 shows an underconstrained cluster configuration that is missing one constraint. The corresponding graphs for a cluster with two and three elements are shown on the right. A decomposition sequence of the set of clusters,

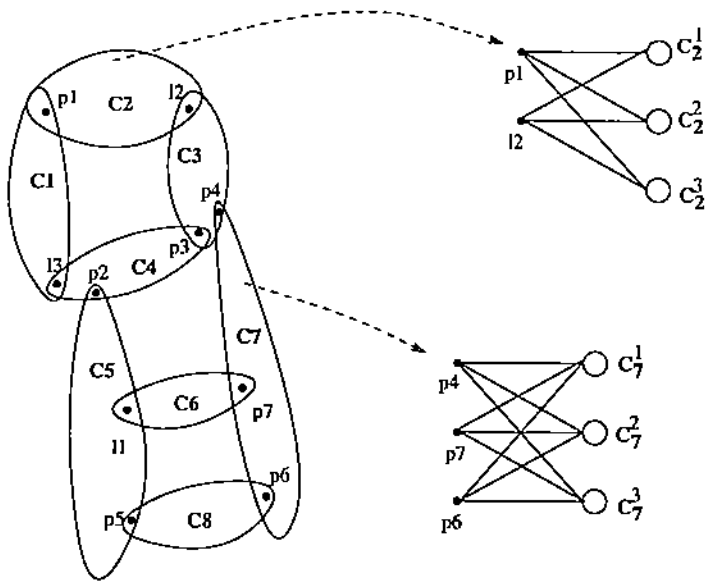


Figure 15: (left) A cluster configuration that is derived from an underconstrained system of geometric constraints, (right) the graphs representing a cluster with two and a cluster with three geometric elements.

is the following:

$$\begin{aligned}
\{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\} &\rightarrow \\
\{\{C_1\}, \{C_2\}, \{C_3, C_4, C_5, C_6, C_7, C_8\}\} &\rightarrow \\
\{\{C_1\}, \{C_2\}, \{C_3\}, \{C_4\}, \{C_5, C_6, C_7, C_8\}\} &\rightarrow \\
\{\{C_1\}, \{C_2\}, \{C_3\}, \{C_4\}, \{C_5\}, \{C_6\}, \{C_7 C_8\}\} &\rightarrow \\
\{\{C_1\}, \{C_2\}, \{C_3\}, \{C_4\}, \{C_5\}, \{C_6\}, \{C_7\}, \{C_8\}\}
\end{aligned}$$

The configuration is solvable, and the last decomposition is a type (b) decomposition and means that we have to add a cluster (constraint) between $p_5$ and $p_7$ or alternatively, between $p_5$ and $p_4$.

In Figure 16 we see a decomposition into split components of the graph $G_D$ for the cluster configuration of Figure 15. The edges inside the clusters are omitted and they are as in Figure 15 (left). The existence of the triangle,

$$\{\{p_1, l_2, l_3\}, \{(p_1, l_2), (l_2, l_3), (l_3, p_1)\}\}$$

means that there is a decomposition of the cluster in three sets that pairwise share a single geometric element, and that one of them represents a point.

The first step of the algorithm takes time $O(|E_D| + |V_D|)$ [33]. Finding the split components each time the second step is executed takes time $O(|E_D| +$

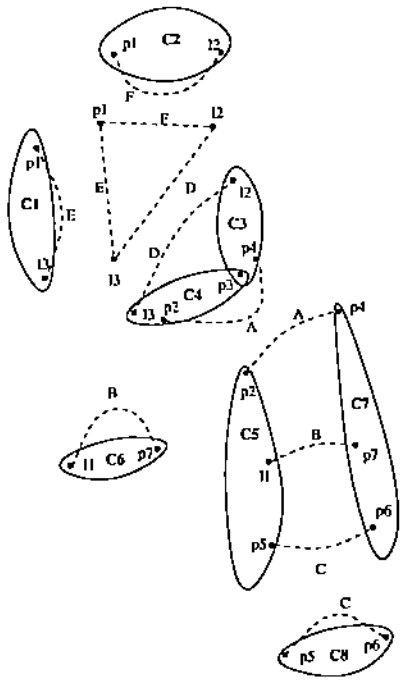


Figure 16: A decomposition in split components of the graph that corresponds to the cluster configuration of the previous figure. The edges inside each cluster are omitted, only the virtual bonds are shown.

$|V_D|$) [15]. The number of times that we need to decompose the components is bounded by $|V_D|$, since each time we decompose the graph, the number of cluster nodes at each component is reduced by one.

Thus the algorithm takes at most $O((|E|+|V|)^2)$ time. If we assume that the number of edges $|E|= O(n)$ then the decomposition analysis of the algorithm takes time $O(n^2)$.

# 5 Extending the Scope of the Analysis Phase

We present two extensions of the scope of the analysis phase. Both extensions are incorporated in the reduction analysis, do not affect the overall time complexity and can be used also in conjunction with the methods of Section 3.3 for solving consistently overconstrained problems.

## 5.1 Extending the Repertoire of Reductions

The reduction analysis can be extended to include more complicated reductions, as the one depicted in Figure 17 for which the constructions are presented in [4]. We replace the DFS of step 3 of the reduction analysis algorithm (Section 3.2), for finding the triangles in graph $G$, with a BFS. The BFS begins from every cluster node of the bipartite graph $H$, to find a 4-cycle or a 6-cycle. If neither can be found, we search for an 8-cycle with exactly 3 incoming edges at the cluster-node of level 4; see also Figure 18. We modify step 4 of the reduction analysis algorithm analogously.

An 8-cycle with exactly 3 incoming edges corresponds to the cluster configuration of Figure 17. In figure 18 we have started a BFS from $C_1$ and at level 4 we detect that the node that corresponds to cluster $C_2$ has exactly 3 incoming edges. Recall that all geometry nodes $g_1, \ldots g_6$, and cluster nodes $C_3$, $C_4$ and $C_5$ have only one incoming edge, since there are no 6-cycles and no 4-cycles by assumption. However, having more than 3 incoming edges to a cluster node of level 4 means that we have an overconstrained configuration. In that case, we can either return the overconstrained subgraph or treat it as a consistently overconstrained configuration by taking into account only three incoming edges, merge the 5 clusters into one and check for consistency using the remaining clusters. Although step 3 is now computationally more expensive, the overall worst-case time complexity remains $O((|E|+|V|)^2)$.

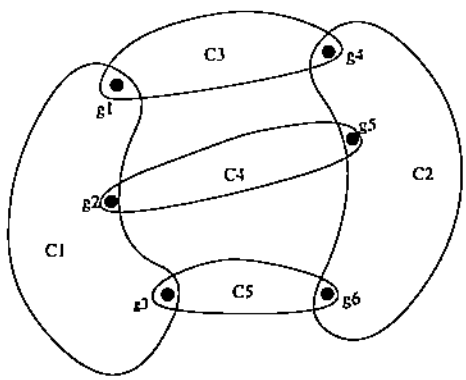


Figure 17: A well-constrained cluster configuration that cannot be solved by the basic method

## 5.2 Incorporating Angle Derivations

By exploiting geometric theorems, we perform certain graph transformations that may convert a constraint graph that is unsolvable by our basic method into an equivalent, solvable constraint graph.

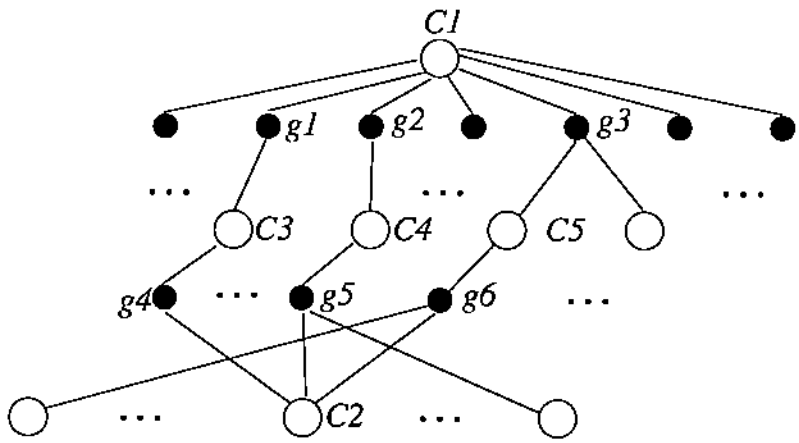


Figure 18: Detecting the cluster configuration of the previous figure during the reduction analysis, by a BFS rooted at $C_1$.

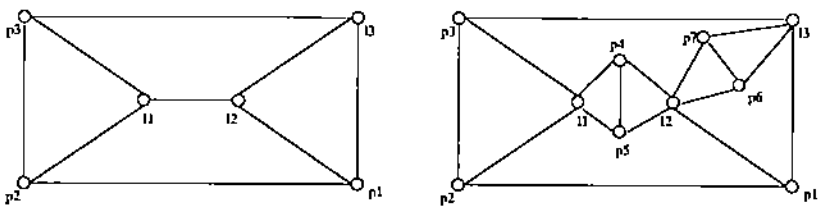

Figure 19: Two constraint configurations that can be solved only if we utilize angle derivations.

A powerful graph transformation rule can be based on the following simple observation: if we have three lines $l_1$, $l_2$, and $l_3$ and the angles $\alpha(l_1, l_2)$ and $\beta(l_2, l_3)$, then the angle between $\gamma(l_1, l_3)$ must be $\pi - (\alpha + \beta)$. This implies that we can substitute for any tree representing $n$ lines and $n-1$ angle constraints between them, by any spanning tree of the complete graph with those $n$ nodes.

In Figure 19 (left) we see a constraint graph that is not solvable by our basic reduction analysis, but it becomes solvable if we replace edge $(l_1, l_2)$ with edge $(l_1, l_3)$ applying the graph transformation rule described above. This rule can be generalized to apply in cases where there are no explicit angles initially between lines, yet implicit angle constraints arise later as a consequence of cluster merging. Figure 19 (right) shows an example.

This graph transformation can be implemented efficiently as follows. We partition all nodes that correspond to line geometries into a number of angle classes. Two vertices belong to the same *angle class* if and only if the angle between them is known. The binary relation *known angle between two lines* is reflexive, transitive and symmetric. We can find the angle classes formed by the initial angle constraints in time $O(|E| + |V|)$, maintaining with each line geometry a bidirectional pointer to its angle class, which is initially set to contain only the line itself. Then we traverse the original constraint graph (e.g. by a DFS), and every time that we find an edge between two lines representing an angle constraint we merge the two corresponding angle classes. All lines in the same cluster belong to the same angle class, we say that the corresponding cluster node belongs to the angle class.

We now modify step 3 of the reduction analysis to check, in addition, whether some angle class triggers a cluster merging. In the graph $H$, this is done by checking which cluster nodes belonging to the angle class have a point vertex in common. This can be done in time $O((|E| + |V|)^2)$. When clusters are merged, the associated angle classes are also merged, and the links to and from the clusters and the corresponding line vertices are updated in time $O(|E| + |V|)$. Finally when we perform a BFS from a cluster vertex upon merging three clusters, we also check whether the associated angle class triggers any further cluster merging, also in time $O(|E| + |V|)$. Thus, the overall worst case time complexity remains $O((|E| + |V|)^2)$.

In Figure 19 (left), there is initially only one angle class $\{l_1, l_2, l_3\}$, whereas in Figure 19 (right) there are initially three angle classes $\{l_1\}$, $\{l_2\}$, and $\{l_3\}$. In Figure 20 (left) we have the cluster configuration of the constraint problem of Figure 19 (left) after the first two cluster merges have occurred. We check to see whether the angle class triggers some additional cluster merging and find that $C_2$ and $C_5$ have a common point vertex $p_3$. Thus clusters $C_2$ and $C_5$ can be merged.

In Figure 20 (right) we have the cluster configuration of the constraint problem of Figure 19 (right) after four cluster merges. The two angle classes and

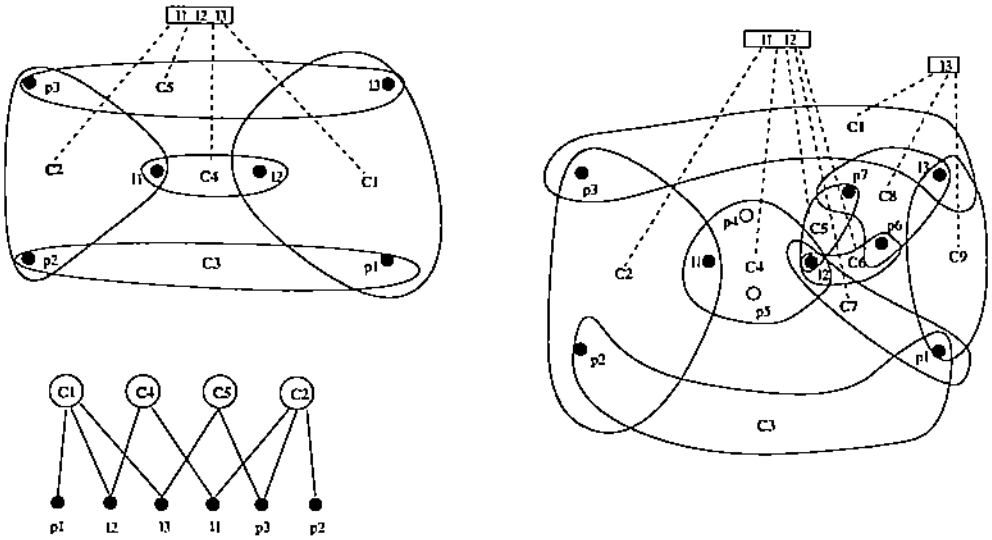


Figure 20: Cluster configurations for the geometric constraint problems of the previous figures after some initial cluster merging has occurred. The corresponding information for the angle classes is also shown.

their links to the clusters are also shown. When clusters $C_5$, $C_6$ and $C_8$ are merged, the corresponding angle classes are also merged. Now the problem becomes similar to the one in Figure 19 (left).

# 6 The Construction Phase of the Algorithm

The graph analysis produces a sequence of instructions to Phase 2 of the constraint solver in which coordinates for the various geometric elements are computed based on the repertoire of construction steps. The geometric construction involves cluster creation and cluster merging. During cluster creation, we place two geometric elements that have a constraint between them with respect to each other. The construction is obvious, since it is a direct interpretation of the equations for distances and angles [8].

In the cluster merging phase we repeat the following: three clusters with three pairwise common geometric elements $g_1$, $g_2$ and $g_3$ are merged into a single cluster, as shown in Figure 21. We place the three common geometric elements, noting that their relative positions are known [8]. Having positioned the shared geometric elements, we translate and rotate the three clusters so that the three geometric elements are in the required location. Note that cluster merging cannot be done when the shared geometric elements are three lines. The constructions for the extension of Section 5.1 is described in [4].

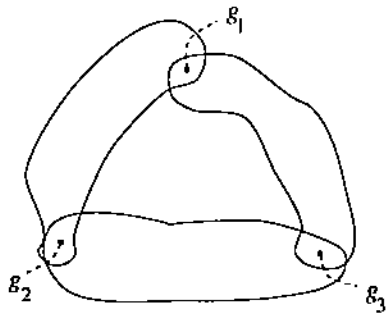


Figure 21: Cluster Merge

## 6.1 Complexity of the Construction Phase

Cluster formation takes time $O(e)$, where $e$ is the number of edges of the constraint graph. The sequence of cluster mergings, derived by the analysis phase, can be done in time $O(e \log_e)$. This is seen as follows.

The rigid motion positioning a cluster, by moving every geometric element of the cluster, requires $O(m)$ steps, where $m$ is the number of geometric elements in the cluster. We consider the largest of the three clusters fixed, and apply the required motion to the two smaller ones. If the largest cluster has $m$ elements, at most $O(m)$ steps are required for the merge. Observe that with this heuristic, the size of the largest cluster is at least $1/3$ the size of the combined clusters. Thus, if a geometric element is in a cluster that is moved, then it becomes part of a new cluster at least twice as large. Consequently, a geometric element cannot be moved more than $\log(r)$ times, where $r$ is the sum of the cardinalities of the clusters and is bounded, by $2e$. Since the number of merge operations is $O(e)$, the construction phase takes at most $O(e \log_e)$ steps.

## 6.2 The Real Solution Problem

We have shown that we can check whether a given geometric constraint problem is solvable by our method in a generic sense in time quadratic in the number of geometric objects. However, computing a real solution for a solvable well-constrained problem will be shown to be NP–hard in a strong combinatorial sense. Although finding a real solution is central to geometric constraint solving, the problem seems to have been neglected in the literature and we are not aware of any results other than a theorem by Hilbert for ruler-constructible configurations. [4, 13]. The following example illustrates the problem of finding a real solution for a constraint configuration.

**Example 4** Given the five points A, B, C, D, E and the following set of geometric constraints:

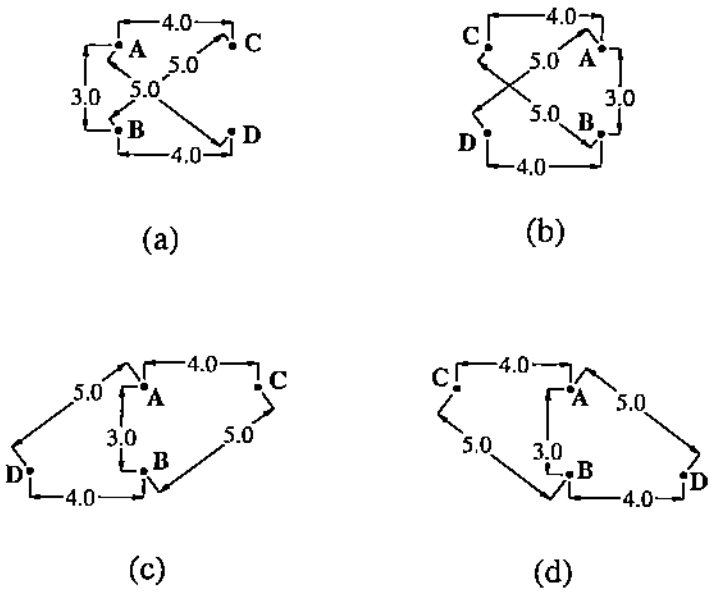


Figure 22: Four discrete solution for placing the four points A, B, C and D with respect to each other

| | |
|---|---|
| A distance 3 from B | |
| C distance 4 from A | C distance 3 from B |
| D distance 3 from A | D distance 4 from B |
| E distance 2 from C | E distance 2 from D |

find a placement of the points that satisfies the above geometric constraints. The placement of the points is with respect to each other, so the derived solution is invariant up to translation and rotation. One placement sequence that satisfies the imposed constraints is the following:

(i) position A and B with respect to each other

(ii) construct C from A and B as the intersection of two circles

(iii) construct D from A and B as the intersection of two circles

(iv) construct E from C and D as the intersection of two circles

Step (i) has only one solution, and for Steps (ii), (iii) and (iv) there are 2 solutions each. Thus, we have eight distinct solution in the complex space. There are four ways in which the points A, B, C, and D may be placed; Figure 22. Solutions (c) and (d) do not extend to a real solution for E, since the circles of step (iv) do not intersect. Solutions (a) and (b) each extend to two real solutions for E, resulting in a total of four real solutions of the constraint problem. □

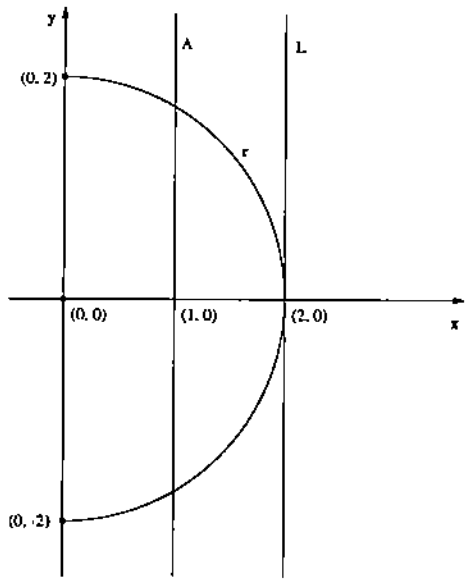


Figure 23: Constructing an instance of the real solution problem

### 6.2.1 NP–hardness

We reduce the following version of 3SAT to the geometric real solution problem.

> *One-in-three monotone 3SAT (1IN3-M3SAT)* [11, 27]: Given a set $U$ of variables, a collection $C$ of clauses over $U$, such that each clause $c \in C$ has three literals, none of them negated. Is there a truth assignment for $U$, such that each clause in $C$ has exactly one true literal?

We will prove that for each instance of 1IN3-M3SAT we can construct, in polynomial time, a solvable geometric constraint problem that has a real solution iff the corresponding instance of 1IN3-M3SAT is satisfiable.

Let $\{x_1, x_2, \ldots x_k\}$ be the set of literals used in $C$. Let $\Lambda$ be the line $x = 1$ and $L$ be the line $x = 2$. For each literal $x_i$, $i = 1 \ldots k$, we construct a point $p_i$ on the $x$–axis at distance 1 from $\Lambda$; see also Figure 23. For each point $p_i$ we can choose independently, one of two solutions, namely $(0, 0)$ or $(2, 0)$, which will correspond to the variable $x_i$ being assigned *false* or *true*, respectively.

For the clause $c = (x_a, x_b, x_c)$, we introduce the points $p'_a, p'_b, p'_c, p_{ab}, p_{bc}, p_{ca}$ and the lines $l_a, l_b, l_c$. The following constraints are imposed:

| | |
|---|---|
| $p_i$ on $x$–axis | $p_i$ distance 1 from $\Lambda$ |
| $p_a$ distance 0 from $l_a$ | $l_a$ angle 0 with the $y$-axis |
| $p_b$ distance 0 from $l_b$ | $l_b$ angle 0 with the $y$-axis |
| $p_c$ distance 0 from $l_c$ | $l_c$ angle 0 with the $y$-axis |

| | |
|---|---|
| $p'_a$ distance 2 from $O(0,0)$ | $p'_a$ distance 0 from $l_a$ |
| $p'_b$ distance 2 from $O(0,0)$ | $p'_b$ distance 0 from $l_b$ |
| $p'_c$ distance 2 from $O(0,0)$ | $p'_c$ distance 0 from $l_c$ |
| $p_{ab}$ distance 3 from $p'_a$ | $p_{ab}$ distance 2 from $p'_b$ |
| $p_{bc}$ distance 3 from $p'_b$ | $p_{bc}$ distance 2 from $p'_c$ |
| $p_{ca}$ distance 3 from $p'_c$ | $p_{ca}$ distance 2 from $p'_a$ |

Thus, the lines $l_u$ are parallel to the $y$-axis through $p'_u$. The points $p'_u$ are at distance 2 from the points $p_u$. Note that to each point $p_u$ there correspond several $p'_u$, one for each occurrence of $x_u$ in a clause of $C$.

**Lemma 3** There is a truth assigmment for $U$ with exactly one true literal in each clause iff the system of geometric constraints described above has a real solution.

**Proof**

$\Longrightarrow$: Assume that there is a truth assignment for $U$ with exactly one true literal in each clause. We set $p_m = (2,0)$ whenever $x_m$ =*true*, and the value $p_n = (0,0)$ whenever $x_n$ =*false* for a solution of $U$. Since a solution of $U$ requires that exactly one literal in each clause $c = (x_a, x_b, x_c)$ is true, exactly one of the points $p_a$, $p_b$, or $p_c$ is at (2,0), say $p_b$. Then $l_b$ coincides with $L$ and $l_a$ and $l_c$ coincide with the $y$-axis. Consequently, $p'_b$ has a unique solution: (2, 0), and for each of the points $p'_a$ and $p'_c$ there are two solutions, namely: $(0,2)$ or $(0,-2)$. If we choose the value $(0,2)$ for $p'_a$ and the value $(0,-2)$ for $p'_c$ we see that the points $p_{ab}$, $p_{bc}$, and $p_{ca}$ are real. Thus the geometric constraint problem has a real solution.

$\Longleftarrow$: Clearly, every point $p_i$ must be at the origin or at $(2,0)$, and these position can be interpreted as a truth assignment to the variables $x_i$ of the formula. Assume that $(2,0)$ is interpreted as *true*, and there is no truth assigment for $U$ with exactly one true literal in each clause. Let $c = (x_a, x_b, x_c)$ be a clause that does not have exactly one *true* literal. We distinguish two cases:
*(i)* $x_a = x_b = x_c$ =true, *or* $x_a = x_b = x_c$ =false*:* Then the lines $l_a$, $l_b$, and $l_c$ coincide, hence so must at least two of the points $p'_a$, $p'_b$ or $p'_c$, say $p'_a$ and $p'_b$. But then we cannot place $p_{ab}$ with real coordinates.
*(ii)* $x_a = x_b$ =true*:* Then the lines $l_a$ and $l_b$ coincide with $L$, hence $p'_a = p'_b = (2,0)$. Again, $p_{ab}$ cannot have real coordinates.

Now assume that $(0,0)$ is interpreted as *true*. By the above argument, there is a real solution of the constraint problem only if every clause of $U$ has exactly two *true* literals. But then the complement assignment solves $U$. Hence there cannot be a real solution of the constraint problem.□

**Proposition 1** The problem of finding a real solution of a system of geometric constraints solvable by a constructive method is NP–hard.

The problem may not even be in NP, since it involves real arithmetic. However, it naturally falls in $DNP_{\mathbb{R}}$(see e.g., [3, 6]), where the guess is polynomial to the size of the input but the checking procedure needs infinite precision to give an answer in polynomial time.

### 6.2.2 Discussion

We have proved that the problem of finding a real solution for a system of geometric constraints that is solvable by our constructive method is NP–hard in a strong combinatorial sense. From the nature of the proof we conclude that several specialization of this problem are also NP–hard. In particular, all subproblems derived by imposing any combination of the following restrictions are also NP–hard:

the set of geometric objects consists only of lines and points.

the set of geometric constraints consists only of distances and angles.

the domain of the value of the geometric constraints is $\{0, 1, 2\}$.

the geometric problem is well-constrained.

Furthermore, if we restrict the geometric objects to be points only and the geometric constraints to be distances only, then we can prove that the problem remains NP–hard.

When the geometric problem is underconstrained, we have argued in this paper that additional constraints should be added to make the problem well-constrained and solvable. It remains to be investigated whether the addition of constraints can facilitate finding real solutions.

# 7 Remarks

We have presented a quadratic algorithm that solves a system of geometric constraints using a graph constructive approach. The bottom-up analysis algorithm of [4] has been augmented with a top-down decomposition algorithm and so not only solves consistently overconstrained problems but underconstrained problems as well. Both over- and underconstrained problems occur in practice and it is important to handle them.

It should be possible to speed up the analysis and the decomposition to take only $O(n \log(n))$ time for constraint graphs with $O(n)$ edges. Such an improvement would be important in applications where large constraint problems are not uncommon.

## References


[1] S. Ait-Aoudia, R. Jegou, and D. Michelucci. Reduction of constraint systems. In *Proc. Compugraphics*, pages 83–92, Alvor, Portugal, 1993.

[2] B. Aldefeld. Variation of geometries based on a geometric-reasoning method. *Computer Aided Design*, 20(3):117–126, April 1988.

[3] L. Blum, M. Schub, and S. Smale. On a theory of computation and complexity over the real numbers: NP–completeness, recursive functions and universal machines. *Bulletin (New Series) of the American Mathematical Society*, 21:1–46, 1989.

[4] W. Bouma, I. Fudos, C. M. Hoffmann, J. Cai, and R. Paige. A Geometric Constraint Solver. *Computer Aided Design*, 27(6):487–501, June 1995. Also available through Mosaic, http://www.cs.purdue.edu/people/fudos.

[5] Beat Bruderlin. Constructing Three-Dimensional Geometric Objects Defined by Constraints. In *Workshop on Interactive 3D Graphics*, pages 111–129. ACM, October 23-24 1986.

[6] F. Cucker and M. Matamala. On digital nondeterminism. preprint.

[7] Andreas W. M. Dress and T. F. Havel. Shortest-Path Problems and Molecular Conformation. *Discrete Appl. Math.*, 19(3):129–144, March 1988.

[8] I. Fudos and C. M. Hoffmann. Correctness Proof of a Geometric Constraint solver. *International Journal of Computational Geometry & Applications*, 1995. Also available through Mosaic, http://www.cs.purdue.edu/people/fudos.

[9] Ioannis Fudos. Editable Representations for 2D Geometric Design. Master's thesis, Dept of Computer Sciences, Purdue University, December 1993. Available through Mosaic, http://www.cs.purdue.edu/people/fudos.

[10] Ioannis Fudos. *Constraint Solving for Computer Aided Design*. PhD thesis, Department of Computer Sciences, Purdue University, August 1995.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Co., 1979.

[12] J. Gosling. Algebraic Constraints. Technical Report CMU-CS-83-132, CMU, 1983.

[13] D. Hilbert. *Grundlagen der Geometrie*. B. G. Teubner, Stuttgart, 1956.

[14] C. M. Hoffmann and R. Juan. EREP : An Editable High-Level Representation for Geometric Design and Analysis. In P. Wilson, M. Wozny, and M. Pratt, editors, *Geometric and Product Modeling*. North Holland, 1993.

[15] J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM J. COMPUT.*, 2(3):135–158, 1973.

[16] H. Imai. On combinatorial structures of line drawings of polyhedra. *Discrete and applied Mathematics*, 10:79, 1985.

[17] Alon Itai and Michael Rodeh. Finding a Minimum Circuit in a Graph. *SIAM J. COMPUT.*, 7(4):413–423, 1978.

[18] R. Juan-Arinyo and Antoni Soto. A rule-constructive geometric constraint solver. Technical Report LSI-95-25-R, Universitat Politecnica de Catalunya, 1995.

[19] Glen A. Kramer. *Solving geometric constraints systems : a case study in kinematics*. MIT Press, Cambridge, Mass., 1992. ISBN:0262111640.

[20] Robert Light and David Gossard. Modification of geometric models through variational geometry. *Computer Aided Design*, 14(4):209–214, July 1982.

[21] S. Mac Lane. A structural characterization of planar combinatorial graphs. *Duke Math. J.*, 3:460–472, 1937.

[22] B. Monien. The Complexity of Determining a Shortest Cycle of Even Length. *Computing*, 31:355–369, 1983.

[23] G. Nelson. Juno, a costraint-based graphics system. In *SIGGRAPH*, pages 235–243, San Francisco, July 22-26 1985. ACM.

[24] J. C. Owen. Algebraic Solution for Geometry from Dimensional Constraints. In *ACM Symp. Found. of Solid Modeling, Austin, TX*, pages 397–407. ACM, 1991.

[25] Dieter Roller. Dimension-Driven geometry in CAD : a Survey. In *Theory and Practice of Geometric Modeling*, pages 509–523. Springer Verlag, 1989.

[26] B. Rosen. Tree-manipulating systems and Church-Rosser theorems. *J. ACM*, 20:160–187, 1973.

[27] T. J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th Ann ACM symp. on Theory of Computing*, pages 216–226, New York, 1978. ACM.

[28] D. Serrano and D. Gossard. Combining mathematical models and geometric models in CAE systems. In *Proc. ASME Computers in Eng. Conf.*, pages 277–284, Chicago, July 1986. ASME.

[29] K. Sugihara. Detection of Structural Inconsistencies in Systems of Equations with Degrees of Freedom and its Applications. *Discrete Applied Mathematics*, 10:297–312, 1985.

[30] Geir Sunde. Specification of shape by dimensions and other geometric constraints. In M. J. Wozny, H. W. McLaughlin, and J. L. Encarnacao, editors, *Geometric modeling for CAD applications*, pages 199–213. North Holland, IFIP, 1988.

[31] I. Sutherland. Sketchpad, a man-machine graphical communication system. In *Proc. of the spring Joint Comp. Conference*, pages 329–345. IFIPS, 1963.

[32] Hirimasa Suzuki, Hidetoshi Ando, and Fumihiko Kimura. Variation of geometries based on a geometric-reasoning method. *Comput. & Graphics*, 14(2):211–224, 1990.

[33] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. COMPUT.*, 1:146–159, 1972.

[34] A. Verroust, F. Schonek, and D. Roller. Rule-oriented method for parameterized computer-aided design. *Computer Aided Design*, 24(3):531–540, October 1992.

[35] Yasushi Yamaguchi and Fumihiko Kimura. A constraint modeling system for variational geometry. In M. J. Wozny, J. U. Turner, and K. Preiss, editors, *Geometric Modeling for Product Engineering*, pages 221–233. Elsevier Science Publishers B.V. (North Holland), 1990.